RC22377 (W0203-059) March 19, 2002
Computer Science

IBM Research Report

Resource Model for Self-Managing Computing Utility Services

Karen Appleby, Tamar Eilam, Liana Fong, Gérman Goldszmidt, Michael Kalantar

IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Submission to DSN-2002 — Workshop on Dependability of E-Commerce Systems

Resource Model for Self-Managing Computing Utility Services

Karen Appleby Tamar Eilam Liana Fong Gérman Goldszmidt Michael Kalantar

IBM T. J. Watson Research Center

30 Saw Mill River Road,

Hawthorne, NY 10532

914-784-7488

914-784-6094

eilamt@us.ibm.com

Abstract

Océano, an IBM Research project, provides management software that enables a Web hosting service to automatically and dynamically relocate Web servers among a set of customer Web sites in response to usage. The Océano approach addresses two problems in current hosting centers: over-provisioning of resources and high manual cost of operation. The goal of the work described in this paper is to generalize the Océano solution so that it can be applied to manage services with different requirements and from different application domains (e.g., Grid or e-business services). We introduce the concept of a self-managing virtual resource: a resource combined with monitoring, problem determination, configuration and allocation policy. Resources are created in a hierarchical fashion from other resources.

Sets of virtual resources can be combined to create services. These services themselves can be viewed as virtual resources used by higher-level services. This approach enables self-managing services to be built in an incremental and modular fashion. We provide illustrative examples inspired by Océano of the model and how it can be applied.

Keywords: self-managing, Internet services, e-commerce, Grid, policy

Submission Category: short work-in-progress paper

Word count: 2777

Declaration: Material must be cleared through author affiliations.

Contact author:

Tamar Eilam

IBM T.J. Watson Research Center

30 Saw Mill River Road,

Hawthorne 10532, NY

914-784-7488

914-784-6094

eilamt@us.ibm.com

1 Introduction

Océano [1] provides management software that enables a Web hosting service to allocate servers to several customer Web sites in response to usage. Océano maintains a pool of servers on a common network infrastructure. Mechanisms are provided that automate the allocation of theses servers to hosted customers' Web sites. When a server is allocated, it may be primed with a fresh copy of an operating system and with customer applications and data; the network is then reconfigured to place the server into the customer's service domain. Performance monitoring data from the server is correlated with data from the other servers resulting in an overall view of the performance of the Web site. The overall performance of the Web site will then drive subsequent cycles of resource allocation or de-allocation.

This basic model in which computing resources can be obtained *on demand* rather than obtaining the maximum required in advance is applicable to services other than Web hosting. For example, a Grid service may offer computing resources on demand by aggregating resources made available from various virtual organizations and by automating their configuration (with respect to other resources). However, detailed requirements to support resources on demand may vary among different types of services. For example:

1. From the viewpoint of a service, different resource abstractions are required. For example, a Grid service might manage servers by their characteristics (memory, disk size, speed, etc). On the other hand, a Web hosting service might manage servers according to their functional usage: a firewall server, or a load balancer server. In both cases, the underlying resources are the physical servers, but the abstractions used vary.
2. Each service might have different policies govern the allocation of resources. In general, all services require additional resources when demand increases. However, the identifiers of overload differ depending on the service. For example, an indication that a Web hosting service needs additional resources might be that the response time (or the ping response time) increases. A video service might identify overload by measures of jitter, and loss rate.
3. Each service might have different policies govern the reaction to resource scarcity. A particular service might have several potential responses for dealing with overload. For example a Web hosting service could not only increase the number of Web servers, but might also reject some portion of requests (e.g., "buy" vs. "browse" requests). In addition to increasing server resources a video serving service might reduce the quality of the video it is currently serving.
4. Resource reclamation differs for different services. For example, a server used by a Web service as a Web server can be reclaimed fairly easily, quickly, and with little impact. To do so, new requests to the server are terminated (via changes to the configuration of a load balancer). Once current requests are completed, the server can be reclaimed. On the other hand, a server being used in a Grid service to execute a long running scientific application with specific server assumptions, cannot be reclaimed without terminating the application (perhaps by causing it to fail) or until the application is finished execution.

5. Different services may want to implement similar resources in different ways. For example, one Web hosting service might build a service using two abstract resources: Web servers and load balancers, while another Web hosting service may implement the service based on three abstract resources: Web servers, load balancers and firewalls.

In order to accommodate these requirements, we describe initial thoughts about how to generalize the Océano resource model. The generalized model creates virtual resources, in a hierarchical fashion, out of other more basic resources (that are possibly not self-managing). Combined with monitoring, problem determination, configuration and allocation policy, the resources become self-managing. Sets of virtual resources can be combined to create services. These services themselves can be viewed as virtual resources used by higher-level services.

Using this model we can build autonomous systems with different requirements and policy in an incremental fashion, reusing more basic self-managing resource components and combining them in a variety of ways.

In the following sections, we will describe the resource model and borrow ideas from the Océano prototype as an illustrative example.

2 Approach

In this section we briefly describe the resource model for building self-managing services. We give examples inspired by our Océano prototype – a self-managing Web-hosting solution – that the proposed model aims at generalizing.

2.1 A Resource Virtualization Model

The resource virtualization model provides multiple levels of abstraction between the actual infrastructure resources and the hosted service. We identify relationships and interdependencies between resources and we use them to build more complex resources in a hierarchical fashion. We introduce the concept of a *self-managing resource:* a resource (possibly an aggregation of sub-resources) that can manage itself by monitoring, correlation, and self-configuration based on a *resource policy*. Resources generate *events* that are consumed by other resources; they receive configuration *requests* and can generate requests to other resources. In the sequel, when clear from the context, we use the terms resource and self-managing resource interchangeably.

Every resource is associated with a set of managing elements that may include:

- Monitoring agents: Monitor a set of metrics and generate *events* (reside on basic resource elements).
- Problem Policy: Receives event and decides on an action according to some *policy rules* in the form of *event? action.* An action can be generated by another event, an external request to some other resource or a request to the resource's configuration policy (see hereafter).
- Configuration Policy: Receives configuration requests and carries out the request possibly by communicating with the configuration policy of the sub-resources.

We identify the following relationships between resources:

- Specialization: A resource can be a special case of another resource. For example, Apache server resource is a special case of a Web server resource.
- Aggregation: A resource can be an aggregation of other resources (termed *sub-resources* of the aggregate). For example, a Web server resource is an aggregation of a machine resource, an operating system resource and a Web server application resource. Note, that a same sub-resource can be contained in different aggregates, for example, a server that is logically partitioned can be used by a set of virtual services.
- Co-location: Some resources must be co-located. For example, an operating system resource and a server. (Note that a "location" should be defined). Some other resources must not be co-located (anti-co-location relation).
- Operational dependence: Some resources require other resources to be in a certain state in order to operate or require information (events) from other resources in order to be re-configured. For example, in some cases a database must be operational when configuring an application resource.

These interdependencies should be taken into account when building complex resources from a set of more basic resources and when defining the Problem Policy and Configuration Policy. In many cases, events from resources with an operational dependency are correlated to determine the root-cause of a problem, and are propagate from sub-resources to containing resources. Also, the co-location relation is important when composing a resource instance from other resources, and the operational dependency is crucial to the definition of the sequence of operations needed to re-configure the system.

As an example of self-managing resources and interrelationships, consider a Web-site service resource that is an aggregation of a set of Web server sub resources and a set of load-balancer sub resources. By correlating events received from the sub-resources the Problem Policy can determine that one of the load-balancers is overloaded. Changing the assignment of Web servers to load-balancers can alleviate the load problem. The Configuration Policy component will orchestrate the change in configuration by communicating (sending requests and receiving replies) with the sub-resources Configuration Policy. For example it can send a request to the Configuration Policy of the overloaded load-balancer to remove one Web-server, wait for the reply, and then send a request to the Configuration Policy of a different load-balancer to add the server. A different scenario is if all

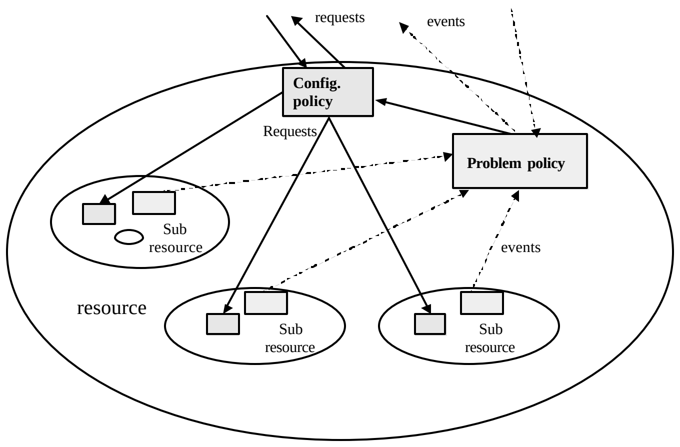

Figure 1: Self-managing resources and interrelationships.

the Web servers are overloaded. In this case, the Policy Component might decide to send a request to the hosting service to provide more Web servers.

This approach allows us to elucidate the configuration and operational complexity of a large set of interdependent components that form a service by breaking it up in a modular hierarchical fashion. It also encourages modular development of self-managing services by re-using a same sub-resource for building different complex resources.

2.2 Configuration Policy and Resources Requests

Configuration Policy components (in short, configuration components) receive configuration requests and carry out all configuration operations on the resource that they manage. Configuration operations are carried out by sending requests to the set of configuration components of the sub-resources in the aggregate. Thus, a configuration component does not have to know the internal details of the sub-resources but it has to know the interdependencies between the sub-resources and the order of operations needed on these sub-resources to carry out a request.

There are three different types of requests: (1) a re-configuration request (e.g., changing the assignment of servers to load balancers, or changing the size of a threads pool in a Web-server), (2) a request to increase a particular sub-resource's capacity, (3) a request to reduce a sub-resource's capacity. Every resource has its own API for configuration requests that depend on the functionality. Requests to increase or reduce capacity have a somehow similar structure that is a Boolean expression over *n*-tuples of resources, properties, and quantities.

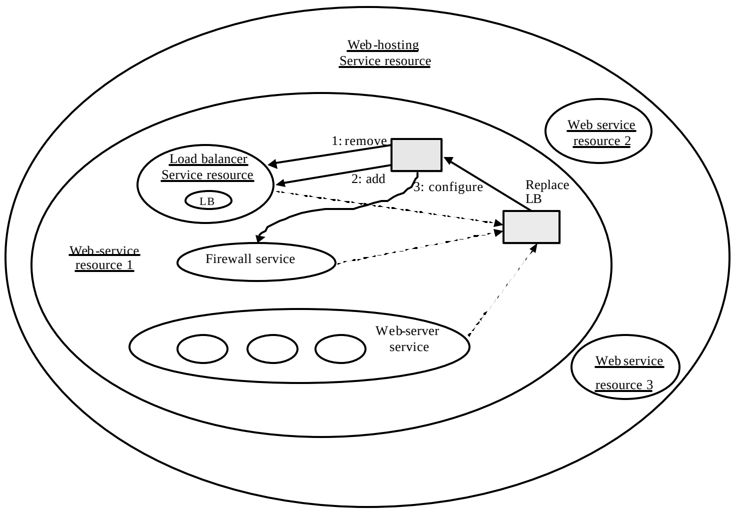

Figure 2: An example that illustrates a Web-site service resource, and the process of replacing a malfunctioning adapter.

The rest of this section is an example inspired by the Océano prototype that illustrates the usage of configuration components and requests. Resources are defined and aggregated as follows. A Web-site service resource is an aggregate of load-balancer service resource, a Web-server resource, and a firewall service resource[1] (see Figure 2). A Web-server resource is a set of basic-Web-server resources each of which is a single Web-server. Suppose that the load balancer service stops functioning as a result of, say, a malfunctioning adapter (see Section 2.3 for a description of the problem

[1] Supporting firewalls in Océano is a work in progress.

determination for this case). The Web-site Problem Policy can decide to replace the load-balancer. It will send a request to the configuration component that will orchestrate the process. Specifically, the Web-site configuration component will ask the load-balancer service to remove the load-balancer, and then to get a new load-balancer. To get a new load-balancer the load balancer service sends a request to a *renting service* (not depicted) that serves also as a *factory* responsible in this case to get an appropriate server and install the load balancer application. The request can be granted or denied according to the renting service policy taking into account availability of resource (e.g., servers, application licenses) and other considerations (beyond the scope of this short paper). If the request is granted the Web-site configuration component has to re-configure other resources in the aggregate appropriately. For example, it sends a request to the firewall to open a set of IP ports (if they are not opened yet).

The sequence of configuration operations that has to be carried out for every case can depend on the set of applications used and the infrastructure configuration. For example, firewalls can be used or not, databases can be used or not. With the proposed model it is relatively easy to modify a self-managing Web-site service to support a different set of applications or configuration by keeping the same structure and modifying only the implementation of some of the configuration components. In a Grid-environment adding a server might include only sending a request to the scheduler service to include the server in the server pool it manages. The Grid service configuration component has to implement this logic, but, it may be able to use the same implementation of other self-managing sub-resources.

2.3 Problem Policy and Monitoring

Each resource accepts a set of events that act as input to problem determination. The resource's problem policy correlates events, queries devices, and analyzes previously opened problems to pinpoint the root cause of each reported event. The output of a resource's problem policy may include the publishing of resource state events, the creation of problem records, or the initiation of corrective actions.

Resources at the lowest levels are directly associated with physical devices. These resources accept external events which originate from the devices themselves. Higher level resources represent progressively more abstract resources or those that are directly dependent on lower resources. Higher level resources accept internal and external events. Internal events are generated by lower level resources and represent aggregations and generalizations of the events that were processed by the policies of lower level resources.

As many dependencies may exist, events are published without regard to who might consume them. It is difficult to know in advance all the resources that will be affected by any given event. In addition maintenance of explicit dependency information is a laborious task. Using this model new resources can be added to the system without explicitly specifying all of the resources it depends on. Only the policies of resources that are interested in the new resource's state and will consume its out-going events need to be modified.

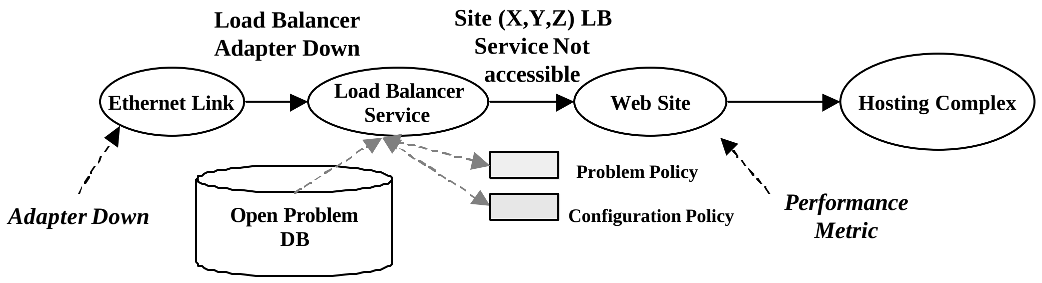

Figure 3: "Adapter Down" Internal and External event propagation

Figure 3 depicts one possible event flow for an "*Adapter Down*" event. The external "*Adapter Down*" event is first processed by the *Ethernet-Link*'s Resource-Policy which will open a medium priority adapter-down problem. It will then look up the type of the reporting device and generate a *resourceType-adapter-down* event which is consumed by the resource of that type, in our case, a *Load-Balancer*. The Load-Balancer-Service Policy determines if the adapter that is down is critical and if so determines which Web-Sites it was performing Load-Balancing for and generates a *Load-Balancing-Service-Not-Accessible* event for the identified sites. In addition a replacement load-balancer can be requested. This processes is initiated by sending a request for a new Load-balancer to the Web-site configuration policy (see Section 2.2); it also escalates the *Adapter-Down* problem and opens an additional *Load-Balancer-Down* problem in the open-problem database. When the Web-Site Resource-Manager receives the *Load-Balancing-Service-Not-Accessible* event it will open a *Web-Site-Down* problem and initiate the SLA violation evaluation process. It should be noted that chain of open problems are linked and can be managed as a set.

Using this model it is possible for each virtual resource to be owned by and created by a different entity. As long as a virtual resource accepts the appropriate input events and takes responsibility for generating the appropriate output events and initiating corrective actions, the virtual resource policy can be written in any language and use any type of logic.

2.4 Related Work

An enormous body of work is related to providing policy driven resource management of systems. We mention only a few of these works that we find most relevant to the discussion in this short paper.

The work [6] proposes a framework for automation of the configuration of Internet services considering cross-component relationship. Resources publish state information that is discovered by other resources and can trigger them to change their configuration correspondingly. The structure is flat in comparison with the hierarchical model that we propose.

The work [5] addresses the challenge of hosting centers from the point of view of energy conservation and management. A set of metrics is monitored to decide when to change the CPU capacity assigned to every customer based on policy.

Mounties [7] provides High Availability service based on a complex resource model that captured also interdependencies between resources.

The Grid community [3][2] mostly focused so far on providing scheduling, match-making, and co-allocation algorithms and protocols that are needed in order to leverage resources distributed over a wide area network. Recently, the data-grid effort [4] identified the need to provide tools for automation of configuration as a part of a Grid infrastructure taking into account complex interdependencies.

3 References

[1] K. Appleby, L. Fong, , G. Goldszmidt, S. Krishnakumar, S. Fakhouri, D. Pazel, J. Pershing, and B. Rochwerger,. "Océano - SLA-based management of a computing utility", *Proc. of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM2001)*, May, 2001

[2] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", Intl. J. Supercomputer Applications, 15(3), 2001.

[3] http://www.gridcomputing.com

[4] http://grid.web.cern.ch/grid/new_home.htm

[5] J. S. Chase, D. C. Anderson, P. N.Thakar, A. M. Vahdat, "Managing Energy and Server Resources in Hosting Centers", *Proceedings of SOSP* 2001.

[6] Todd Poynor, "Automating Infrastructure Composition for Internet Services", *Proceedings of 15th System Administration Conference (LISA 2001)*

[7] S. A. Fakhouri, W. F. Jerome, V. K. Naik, A. Raina, P. Varma, "Active Middleware Services in a Decision Support System for Managing Highly Available Distributed Resources", *Proceedings of Middleware 2000*.